Scheduling independent stochastic tasks deadline and budget constraints

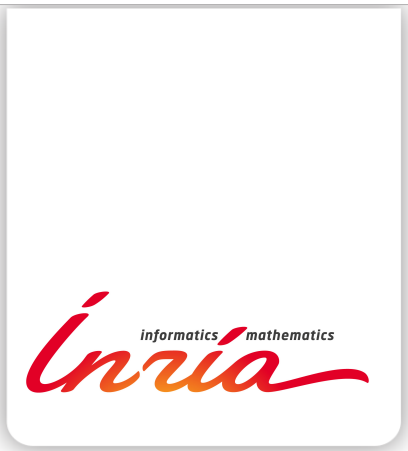
Louis-Claude Canon, Aurélie Kong Win Chang, Yves Robert, Frédéric Vivien

Scheduling independent stochastic tasks deadline and budget constraints

Louis-Claude Canon, Aurélie Kong Win Chang, Yves Robert, Frédéric Vivien

RESEARCH REPORT
N° 9178
June 2018
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9178--FR+ENG



Scheduling independent stochastic tasks deadline and budget constraints

Louis-Claude Canon*†, Aurélie Kong Win Chang*, Yves Robert*‡, Frédéric Vivien*

Project-Team ROMA

Research Report n° 9178 — June 2018 — 34 pages

Abstract: This paper discusses scheduling strategies for the problem of maximizing the expected number of tasks that can be executed on a cloud platform within a given budget and under a deadline constraint. The execution times of tasks follow IID probability laws. The main questions are how many processors to enroll and whether and when to interrupt tasks that have been executing for some time. We provide complexity results and an asymptotically optimal strategy for the problem instance with discrete probability distributions and without deadline. We extend the latter strategy for the general case with continuous distributions and a deadline and we design an efficient heuristic which is shown to outperform standard approaches when running simulations for a variety of useful distribution laws.

Key-words: independent tasks, stochastic cost, scheduling, budget, deadline, cloud platform.

* Univ Lyon, CNRS, ENS de Lyon, Inria, Université Claude-Bernard Lyon 1, LIP UMR5668 LYON Cedex 07 France
† FEMTO-ST, Université de Bourgogne Franche-Comté, France
‡ Univ. Tenn. Knoxville, USA

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Ordonnancement de tâches stochastiques indépendantes avec contraintes de budget et d'échéance

Résumé : Ce rapport présente des statégies d'ordonnacement pour le problème suivant: maximiser l'espérance du nombre de tâches indépendantes exécutées sur une plate-forme de type *cloud computing*, avec une double contrainte de budget et de date d'échéance. Les temps d'exécution des tâches suivent une même loi de probabilité, discrète ou continue. Il faut décider combien de processeurs mettre en oeuvre, et quand interrompre les tâches qui se sont déjà exécutées pendant une certaine durée. Nous donnons des résultats de complexité pour l'instance du problème sans échéance et avec distribution discrète, et proposons une stratégie asymptotiquement optimale. Nous étendons cette stratégie au cas général des distributions continues et avec échéance, et définissons une heuristique qui surpasse les approches usuelles pour une vaste gamme de distribiutions usuelles.

Mots-clés : tâches indépendantes, coût stochastique, stochastic cost, ordonnancement, budget, date d'échéance, cloud platform.

1 Introduction

This paper deals with the following problem: given an infinite bag of stochastic tasks, and an infinite set of available Virtual Machines (VMs, or processors[1]), how to successfully execute as many tasks as possible in expectation, under both a budget and a deadline constraint? The execution times of the tasks are IID (independent and identically distributed) random variables that follow a common probability distribution. The amount of budget spent during the execution of a given task is proportional to the length of its execution. At each instant, the scheduler can decide whether to continue the execution (until success) or to interrupt the task and start a new one. Intuitively, the dilemma is the following: (i) continuing execution means spending more budget, and taking the risk of waiting very long until completion, but it capitalizes on the budget already spent for the task; (ii) interrupting the task wastes the budget already spent for the task, but enables starting afresh with a new, hopefully shorter task. Of course there is a big risk here, since the new task could turn out to have an even longer execution than the interrupted one.

In addition to deciding which tasks to interrupt and when, the scheduler must also decide how many processors to enroll (this is the ressource provisioning problem). There is again a trade-off here. On the one hand, enrolling many processors is mandatory when the deadline is small in front of the budget, and it allows us to make better scheduling decisions, because we can dynamically observe many events taking place in parallel[2]. On the other hand, enrolling too many processors increases the risk of having many unfinished tasks when budget runs out and/or when deadline strikes.

This difficult scheduling problem naturally arises with many applications in the context of cloud computing and data mining (see Section 2 for a detailed discussion). Informally, the goal is to extract as much information as possible from some big data set, by launching analysis tasks whose execution time strongly depends upon the nature of the data sample being processed. Not all data sample must be processed, but the larger the number of data samples successfully processed, the more accurate the analysis.

The main contribution of this work are the following:

- We provide a comprehensive set of theoretical results for the problem instance with discrete distributions and no deadline. These results show the difficulty of the general scheduling problem under study, and lay the foundations for its analysis;
- We design an asymptotically optimal scheduling strategy for the above problem instance (discrete distribution, no deadline)
- We design an efficient heuristic, OptRatio, for the general problem.

[1] Throughout the text, we use both terms VM and processor indifferently.

[2] See the examples of Section 4.1 for an illustration.

This heuristic extends the asymptotically optimal scheduling strategy for discrete distributions to continuous ones, and accounts for the deadline constraint by enrolling the adequate number of processors. The heuristic computes a threshold at which tasks should be interrupted, which we compute for a variety of standard probability distributions (exponential, uniform, beta, gamma, inverse-gamma, Weibull, absolute normal, and lognormal)

- We report a set of simulation results for three widely used probability distributions (exponential, uniform, and lognormal) that demonstrate both the superiority of the OptRatio heuristic over other approaches, and its robustness in front of short deadlines.

The rest of the paper is organized as follows. Section 2 surveys related work. We detail the framework and the objective in Section 3. We deal with discrete probability distributions for task execution times in Section 4; in this section, we establish several complexity results and design an asymptotically optimal scheduling policy. In Section 5, we introduce several heuristics for continuous probability distributions that decide when to interrupt tasks based upon various criteria, one of them being based on an extension of the asymptotically optimal scheduling policy for discrete distributions. We compare these heuristics in Section 5.2, assessing their performance for three widely used distributions, exponential, uniform, and lognormal. Finally, we provide concluding remarks and directions for future work in Section 6.

2 Related work

This work falls under the scope of cloud computing since it targets the execution of sets of independent tasks on a cloud platform under a deadline and a budget constraints. However, because we do not assume to know in advance the execution time of tasks (we are in a *non-clairvoyant* setting), this work is also closely related to the scheduling of bags of tasks. We survey both topics in Sections 2.1 and 2.2. Finally, in Section 2.3, we survey task models that are closely related to our model.

2.1 Cloud computing

There exists a huge literature on cloud computing, and several surveys review this collection of work [4, 33, 34]. Singh and Chana published a recent survey devoted solely to cloud ressource provisioning [33], that is, the decision of which resources should be enrolled to perform the computations. Resource provisioning is often a separate phase from resource scheduling. Resource scheduling decides which computations should be processed by each of the enrolled resources and in which order they should be performed.

Resource provisioning and scheduling are key steps to the efficient execution of workflows on cloud platforms. The multi-objective scheduling

problem that consists in meeting deadlines and either respecting a budget or minimizing the cost (or energy) has been extensively studied for deterministic workflows [1,3,6,7,12,16,24,25,37], but has received much less attention in a stochastic context. Indeed, most of the studies assume a *clairvoyant* setting: the resource provisioning and task scheduling mechanisms know in advance, and accurately, the execution time of all tasks. A handful of additional studies also consider that tasks may fail [23, 32]. Among these articles, Poola et al. [32] differ as they assume that tasks have uncertain execution times. However, they assume they know these execution times with a rather good accuracy (the standard deviation of the uncertainty is 10% of the expected execution time). They are thus dealing with uncertainties rather than a true non-clairvoyant setting. The work in [8] targets stochastic tasks but is limited to taking static decisions (no task interruption).

Some works are limited to a particular type of application like MapReduce [19, 35]. For instance, Tian and Chen [35] consider MapReduce programs and can either minimize the financial cost while matching a deadline or minimize the execution time while enforcing a given budget.

2.2 Bags of tasks

A bag of tasks is an application comprising a set of independent tasks sharing some common characteristics: either all tasks have the same execution time or they are instances coming from a same distribution. Several works devoted to bag-of-tasks processing explicitly target cloud computing [17,31]. Some of them consider the classical clairvoyant model [17] (while [10] targets a non-clairvoyant setting). A group of authors including A.-M. Oprescu and Th. Kielmann have published several studies focusing on budget-constrained makespan minimization in a non clairvoyant settings [29–31]. They do not assume to know the distribution of execution times but try to learn it on the fly [29,30]. This work differs from ours as these authors do not consider deadlines. For instance, in [31], the objective is to try to complete all tasks, possibly using replication on faster machines, and, in case the proposed solution fails to achieve this goal, to complete as many tasks as possible. The implied assumption is that all tasks can be completed within the budget. We implicitly assume the opposite: there are too many tasks to complete all of them by the deadline, and therefore we attempt to complete as many as possible; we avoid replication, which would be a waste of ressources.

Vecchiola et al. [36] consider a single application comprising independent tasks with deadlines but without any budget constraints. In their model tasks are supposed to have different execution times but they only consider the average execution time of tasks rather than its probability distribution (this is left for future work). Moreover, they do not report on the amount of deadline violations; their contribution is therefore hard to assess. Mao et al. [26] consider both deadline and budget constrained provisioning and

assume they know the tasks execution times up to some small variation (the largest standard deviation of a task execution time is at most 20% of its expected execution time). Hence, this work is more related to scheduling under uncertainties than to non-clairvoyant scheduling.

2.3 Task model

Our task model assumes that some tasks may not be executed. This model is very closely related to *imprecise computations* [2, 11, 22], particularly in the context of real-time computations. In imprecise computations, it is not necessary for all tasks to be completely processed to obtain a meaningful result. Most often, tasks in imprecise computations are divided into a mandatory and an optional part: our work then perfectly corresponds to the optimization of the processing of the optional parts. Among domains where tasks may have optional parts (or some tasks may be entirely optionals), one can cite recognition and mining applications [27], robotic systems [18], speech processing [14], and [21] also cites multimedia processing, planning and artificial intelligence, and database systems. Our task model also corresponds to the overload case of [5] where jobs can be *skipped* or *aborted*. Another, related model, is that of *anytime tasks* [20] where a task can be interrupted at any time, with the assumption that the longer the running, the higher the quality of its output. Such a model requires a function relating the time spent to a notion of reward. Finally, we note that the general problem related to interrupting tasks falls into the scope of optimal stopping, the theory which consists in selecting a date to take an action, in order to optimize a reward [15].

Altogether, the present study appears to be unique because it is non-clairvoyant and assumes an overall deadline in addition to a budget constraint.

3 Problem definition

This section details the framework and scheduling objective.

Tasks We aim at scheduling a set of independent tasks whose execution times are IID (independent and identically distributed) random variables. The common probability distribution of the execution time is denoted as $\mathcal{D}$. We consider both discrete and continuous distributions in this work. Discrete distributions are used to better understand the problem. Continuous distributions are those typically used in the literature, namely exponential, uniform, and lognormal distributions.

Platform The execution platform is composed of identical VMs, or processors. Without lost of generality, we assume unit speed and unit cost for

each VM, and we scale the task execution times when we aim at changing granularity. Execution time and budget are expressed in seconds. There is an unlimited number of VMs that can be launched by the user.

Constraints and optimization objective The user has a limited budget b and an execution deadline d. The optimization problem is to maximize the expectation of the number of tasks that can be completed until: (i) the deadline is reached; and (ii) the totality of the budget is spent. More precisely:

- The scheduler decides how many VMs to launch and which VMs to stop at each second;
- Each VM executes a task as soon as it is started;
- Each VM is interrupted as soon as the deadline or the budget is exceeded, whichever comes first;
- Each task can be deleted by the scheduler at any second before completion;
- The execution of each task is non-preemptive, unless specified otherwise. In a non-preemptive execution, interrupted tasks cannot be relaunched, and the time/budget spent computing until interruption is completely lost. On the contrary, in a preemptive execution, a task can be interrupted temporarily (e.g., for the execution of another task, or until some event on another VM) and resumed later on.

4 Discrete distributions

This section provides theoretical results when execution times follow a discrete probability distribution $\mathcal{D} = \{(p_i, w_i)\}_{1 \leq i \leq k}$. There are k possible execution times $w_1 < w_2 < \cdots < w_k$ (expressed in seconds) and a task has an execution time w_i with probability p_i, where $\sum_{i=1}^{k} p_i = 1$. The w_i are also called *thresholds*, because they represent instants at which we should take decisions: if the current task did not complete successfully, then either we continue its execution (if the remaining budget allows for it), or we interrupt the task and start a new one. Of course the discrete distribution of the thresholds is somewhat artificial: in practice, we have continuous distributions for the execution times of the tasks. With continuous distributions, at any instant, we do not know for sure that the task will continue executing until some fixed delay. On the contrary with discrete distributions, we know that the execution will continue (at least) until the next threshold. However, any continuous distribution can be approximated by a discrete distribution, and the more threshold values, the more accurate the approximation. In Section 5, we use the results obtained for discrete distributions to design efficient strategies for continuous distributions.

In this section, we further assume that there is no scheduling deadline

d, or equivalently, that the deadline is equal to the budget: $d = b$. We re-introduce deadlines when dealing with continuous distributions in Section 5.

To help the reader apprehend the difficulty of the problem, we start with an example in Section 4.1. We discuss problem complexity without deadline in Section 4.2, providing pseudo-polynomial optimal algorithms and comparing three scenarios: sequential, sequential with preemption, and parallel. Then in Section 4.3, we focus on cases where the budget is large and design an asymptotically optimal strategy. This strategy determines the optimal threshold at which to interrupt all yet unsuccessful tasks. This result will be key to the design of a very efficient heuristic for continuous distributions in Section 5.1.

4.1 Example

We consider the following example with $k = 3$ thresholds: $\mathcal{D} = \{(0.4, 2), (0.15, 3), (0.45, 7)\}$. In other words, with a probability of 40% the execution time of a task is 2 seconds, with a probability of 15% it is 3 seconds, and with a probability of 45% it is 7 seconds. We assume that we have a total budget $b = 6$ (and recall that there is no deadline, or equivalently $d = 6$). Because $b = 6 < w_3 = 7$, no task will ever be executed up to its third threshold. We first define and evaluate the optimal policy with a single processor. Then, we exhibit a policy for two processors that achieves a better performance.

With a single processor Let $E(b)$ denote the optimal expected number of completed tasks when the total budget is equal to b. To define the optimal policy for a budget of 6, we first compute $E(b)$ for the lower values of b that will appear recursively in the expression of $E(6)$.

- $E(1) = 0$, because $w_1 = 2$.
- $E(2) = p_1 \times 1 + (p_2 + p_3) \times 0 = 0.4$: when the budget is equal to 2, the only thing we can do is run the task for two units of time and check whether it completed, which happens with probability p_1. Otherwise, no task is completed.
- $E(3) = (p_1 + p_2) \times 1 + p_3 \times 0 = 0.55$. Once again, we execute the task for two units of time. If it has not succeeded it would be pointless to kill it because the remaining budget is 1 and $E(1) = 0$ (and if it has succeeded we cannot take advantage of the remaining budget). Hence, if the task has not completed after two units of time, we continue its computation for the remaining unit of time and check whether it has succeeded.
- $E(4) = \max\{p_1 + E(2), p_1(1 + E(2)) + p_2(1 + E(1)) + p_3(0 + E(1))\} = 2p_1 = 0.8$. Here, two policies can be envisioned. Either, we decide to

kill the first task if it has not completed by time 2 or, if it has not completed, we let it continue up to time 3 where we kill it if it has not completed (we do not have the budget to let it run up to w_3). In the second case, we distinguish two sub-cases depending on the acutal task duration. The reasoning will be the same for $E(6)$.

- $E(6) = \max\{p_1+E(4), p_1(1+E(4))+p_2(1+E(3))\} = 3p_1 = 1.2$. Once again, two policies can be envisionned. Either, we decide to kill the first task if it has not completed by time 2 or, if it has not completed, we let it pursue up to time 3 where we kill it if it has not completed (we do not have the budget to let it run up to w_3).

Therefore, the optimal expectation with a single processor is to complete 1.2 tasks. The principle used to design the optimal policy will be generalized to obtain Algorithm 1.

With two processors We consider the following policy:

- we start two tasks in parallel;
- if none of them completes by time 2, we let them run up to time 3;
- otherwise, we kill at time 2 any not-yet completed task and start a new task instead.

The following table displays the expected number of completed tasks for each case of execution time of the two tasks initially started:

	w_1	w_2	w_3
w_1	$2+p_1$	$1+p_1$	$1+p_1$
w_2	$1+p_1$	2	1
w_3	$1+p_1$	1	0

For instance, the square at the intersection of the column w_1 and the row w_2 corresponds to the case where the task on the first processor completes in two units of time, where the task on the second processor would have needed 3 units of time. Because of our policy, this second task is killed and at time 2 and we have completed a single task. There remain 2 units of time and we start a third task, which will complete in this budget with probability p_1. Therefore, the total expected number of completed task in this configuration is $1+p_1$, and this configuration happens with probability p_1p_2.

The total expected number of completed tasks is:

$$E' = p_1^2(2+p_1) + 2p_1(p_2+p_3)(1+p_1) + 2p_2^2 + 2p_2p_3 = 1.236.$$

Therefore, this two-processor policy is more efficient than the optimal single processor policy! Even in the absence of deadline parallelism may help to achieve better performance.

This example helps comprehend the difficulty of the scheduling problem under study. The reader may feel frustrated that in the above example, the performance is only improved by 3%. In fact, one of the conclusions of our work is that, in the absence of deadlines, using several processors only marginally improves performance.

4.2 Complexity results

This section is the only one in the paper where we allow preemption. We compare the performance of sequential scheduling, without or with preemption, to that of parallel scheduling, for the problem instance without deadline.

We first present optimal algorithms to solve in pseudo-polynomial time the sequential case without preemption (Algorithm 1) and with preemption (Algorithm 2), as well as an exponential algorithm to solve the parallel case (Algorithm 3). We then show (Lemma 4) that the performance of the first two algorithms bound the performance of the optimal parallel algorithm.

Algorithm 1 is a dynamic programming algorithm that computes in pseudo-polynomial time the expected number of tasks that can be completed on a single processor (without preemption) for a given budget. To ease its writing (and that of Algorithm 2 for the case with preemption) we choose to present it as a recursive algorithm without memoization. Nevertheless, it can easily be transformed into a classical dynamic programming algorithm.

Lemma 1. *Algorithm 1 computes the optimal expected number of tasks that can be completed on a single processor (without preemption) for a given budget b in time $O(kb)$.*

Proof. The main property presiding to the design of Algorithms 1 and 2 is that the only times at which knowledge is gained is when the execution time of a task reaches one of its k thresholds w_1, ..., w_k. (Note that, by definition, a task can only complete at one of these thresholds.) Therefore, it can never be beneficial to stop a non-completed task when its execution time is not equal to a threshold. Therefore, without loss of generality, we focus on algorithms that kill non-completed tasks only at threshold times. Then, the only decision that such an algorithm can take is, when a task reaches a threshold without completing, whether to kill it and start a new task, or continue its execution until the next threshold, where either the task will succeed or a new decision will have to be taken. This is exactly what Algorithm 1 encodes. This algorithm contains at most kb different calls to the function *SeqSched*; hence, the complexity. □

Algorithm 1: Dynamic programming algorithm to compute the optimal expected number of tasks completed within the budget b on a single processor *without* preemption.

Function `SeqSched`(β, s)
  Data: The budget β
  The threshold s at which the last executed task stopped ($s = 0$ if the execution was sucessful)
  bestExpectation $\leftarrow 0$
  `/* If the budget allows it, we can attempt to start a new task */`
  if $\beta \geq w_1$ **then**
    bestExpectation $\leftarrow$ $p_1(1+\texttt{SeqSched}(\beta - w_1, 0))+(1-p_1)(\texttt{SeqSched}(\beta - w_1, 1))$
  `/* If there was a task preempted at threshold s and if the budget allows it, we can try to continue executing this task */`
  if $s > 0$ *and* $w_{s+1} - w_s \leq \beta$ **then**
    if $s = k - 1$ **then**
      expectation $\leftarrow 1 + \texttt{SeqSched}(\beta - (w_{s+1} - w_s), 0)$
    else
      expectation $\leftarrow$ $\frac{p_{s+1}}{1-\sum_{i=1}^{s} p_s}(1 + \texttt{SeqSched}(\beta - (w_{s+1} - w_s), 0)) +$ $\frac{1-p_{s+1}}{1-\sum_{i=1}^{s} p_s}(\texttt{SeqSched}(\beta - (w_{s+1} - w_s), s + 1))$
    bestExpectation $\leftarrow \max\{$*bestExpectation*, *expectation*$\}$
  return *bestExpectation*
return `SeqSched`($b, 0$)

Algorithm 2 is a generalization of Algorithm 1 to the case with preemption. In this context, algorithms no longer kill non-completed tasks, but can preempt them with the possibility to restart them later (or not). In the writing of this algorithm, when S is an array, the notation "$S + a\mathbb{1}_s$" means "add a to the entry s of array S". Algorithm 2 has a pseudo-polynomial complexity only if the maximum number of thresholds, k, is fixed.

Lemma 2. *Algorithm 2 computes the optimal expected number of tasks that can be completed on a single processor with preemption for a given budget b in time $O\left(\prod_{s=1}^{k-1}\left(1+\frac{b}{w_s}\right)\right)$.*

Proof. The proof of correctness and optimality of Algorithm 2 both come directly from that of Algorithm 1. A task preempted at threshold s was executed for a time w_s and, therefore, there can be at most $\frac{b}{w_s}$ such tasks in an execution. Therefore, there are at most $\prod_{s=1}^{k-1}\left(1+\frac{b}{w_s}\right)$ possible values for the array S (a task always completes when it reaches the threshold k). Hence, the complexity. □

Algorithm 3 computes for parallel machines the optimal expected number of tasks that can be completed within the budget, without premeption. We call *progress* of a task the total execution time so far of that task. Let `ParSchedDecision`(β, T_1, T_2) be the expected number of tasks that can be completed with a budget β, knowing the progress of the tasks in the task sets T_1 and T_2 where 1) tasks belonging to T_1 may be interrupted, 2) tasks belonging to T_2 cannot be interrupted, and 3) if the progress of a task is equal to a threshold, that task did not complete at that threshold. Let `ParSchedJump`(β, T) be the expected number of tasks that can be completed with a budget β, knowing the progress of the tasks in the task set T. Finally, let `ParSchedState`(β, T_1, T_2) be the expected number of tasks that can be completed with a budget β, knowing the progress of the tasks in the task sets T_1 and T_2, where 1) the progress of each task in T_1 is equal to a threshold where the task may have succeeded (we have not yet looked whether this is the case), and 2) the progress of a task in T_2 can only be equal to a threshold if the task failed to complete at that threshold. Intuitively, `ParSchedDecision` specifies whether to continue, stop or start tasks, while `ParSchedJump` advances the progress of the tasks, and `ParSchedState` determines which tasks succeed when a threshold is reached.

Lemma 3. *Algorithm 3 computes the optimal expected number of tasks that can be completed on parallel processors without preemption for a given budget b in time $O((b+k)b^3 w_k^b)$.*

Proof. The proof of correctness and optimality of Algorithm 3 also comes from that of Algorithm 1. Any time a threshold is reached, `ParSchedState` is called and determines which tasks succeed or not. Then, `ParSchedDecision`

Algorithm 2: Dynamic programming algorithm to compute the optimal expected number of tasks completed within the budget b on a single processor *with* preemption.

Function `PSeqSched`(β, S)
  Data: The budget β
  An array S of size k: $S[i]$ is the number of tasks preempted at state i
  bestExpectation $\leftarrow 0$
  `/* If the budget allows it, we can attempt to start a new task */`
  if $\beta \geq w_1$ **then**
    bestExpectation $\leftarrow p_1(1 + \texttt{PSeqSched}(\beta - w_1, S)) + (1 - p_1)(\texttt{PSeqSched}(\beta - w_1, S + \mathbb{1}_1))$
  for $s = 1$ *to* $k - 1$ **do**
    `/* If there was a task preempted at threshold` s `and if the budget allows it, we can try to restart one such task */`
    if $S[s] > 1$ *and* $w_{s+1} - w_s \leq \beta$ **then**
      if $s = k - 1$ **then**
        expectation $\leftarrow (1 + \texttt{PSeqSched}(\beta - w_{s+1}, S - \mathbb{1}_s))$
      else
        expectation $\leftarrow$ $\frac{p_{s+1}}{1-\sum_{i=1}^{s} p_s}(1 + \texttt{PSeqSched}(\beta - w_{s+1}, S - \mathbb{1}_s)) + \frac{1-p_{s+1}}{1-\sum_{i=1}^{s} p_s}(\texttt{PSeqSched}(\beta - w_{s+1}, S - \mathbb{1}_s + \mathbb{1}_{s+1}))$
      bestExpectation $\leftarrow \max\{$*bestExpectation*, *expectation*$\}$
  return *bestExpectation*
Let S be an array of size $k - 1$ with $S[i] = 0$ for all i
return `PSeqSched`(b, S)

decides which tasks to continue and whether new tasks must be started. There are at most $q = \lfloor b/w_1 \rfloor$ concurrent running tasks. Thus, `ParSchedDecision` can be called with $bq^2w_k^q$ different arguments. Each call requires q calls to `ParSchedJump`, which can take bqw_k^q different arguments and takes qk operations. Finally, `ParSchedState` costs less than `ParSchedDecision`. Hence, the time complexity is $b(q+k)q^2w_k^q = O((b+k)b^3w_k^b)$ and the space complexity is $bq^2w_k^q = O(b^3w_k^b)$. □

Relations between problems

Lemma 4 formally states that any algorithm for p processors (using or not preemption) can be simulated on a single processor with preemption. From this property, it immediately follows that the performance of the optimal parallel algorithm on p processors (Algorithm 3) is upper bounded by the performance of Algorithm 2 and lower bounded by the performance of Algorithm 1.

Lemma 4. *Any algorithm designed to be executed on p processors with or without preemption can be simulated on a single processor with preemption with the same performance.*

Proof. Consider any algorithm $\mathcal{A}$ designed to be executed on p processors with preemption. As already stated in the proof of Lemma 1, any meaningful algorithm only takes decisions when a task reaches a threshold. Of course, in a parallel algorithm, a task may be stopped in between two of its thresholds if, at that time, another task reaches one of its own thresholds. On the contrary, no knowledge is gained at a time when no task reaches a threshold, and it is thus suboptimal to kill or to preempt a task at such a time. Without loss of generality, we thus assume that $\mathcal{A}$ only kills or preempts tasks at their thresholds.

Without loss of generality, we also assume that all thresholds are integers (otherwise, we just scale the thresholds). Then, we simulate $\mathcal{A}$ as follows to obtain a sequential algorithm $\mathcal{A}^*$. Assume we have simulated $\mathcal{A}$ from time 0 to t. Then $\mathcal{A}^*$ ran from time 0 to t^* (where $0 \leq t^* \leq t \times p$) and spent the same amount of time processing the very same tasks than $\mathcal{A}$. We now simulate the work of $\mathcal{A}$ for the time-interval $[t; t+1]$. Let P_1, ..., $P_{p'}$ be the $p' \leq p$ processors, numbered arbitrarily, that process some work under $\mathcal{A}$ during the interval $[t; t+1]$. Let T be the task processed by P_i during that time under $\mathcal{A}$. Then $\mathcal{A}^*$ processes T during $[t^* + (i-1); t^* + i]$. $\mathcal{A}^*$ can take this decision because at time t^*, $\mathcal{A}^*$ has processed the exact same work than $\mathcal{A}$ at time t. Therefore, at time $t^* + (i-1)$, $\mathcal{A}^*$ has all the necessary knowledge. At time $t^* + p'$, $\mathcal{A}^*$ has processed the exact same work than $\mathcal{A}$ at time $t+1$ and we can conclude by an immediate induction. □

Algorithm 3: Dynamic programming algorithm to compute the optimal expected number of tasks completed within the budget b in parallel.

Function `ParSchedDecision`(β, T_1, T_2)
  Data: The budget β
  A set T_1: $T_1[i]$ is the progress of a task that may be interrupted
  A set T_2: $T_2[i]$ is the progress of a task that cannot be interrupted
  if $\beta = 0$ **then return** 0
  if $T_1 = \emptyset$ **then**
    $q \leftarrow \lfloor \beta / w_1 \rfloor$
    `/* In addition to the current progressing tasks, we can start new ones */`
    return $\max_{0 \leq i \leq q}$ `ParSchedJump`$(\beta, T_2 \cup \{0\}^i)$
  else
    `/* Task 1 in` T_1 `is either interrupted or not */`
    return $\max($`ParSchedDecision`$(\beta, T_1 \setminus \{T_1[1]\}, T_2)$, `ParSchedDecision`$(\beta, T_1 \setminus \{T_1[1]\}, T_2 \cup \{T_1[1]\}))$

Function `ParSchedJump`(β, T)
  Data: The budget β
  A set T: $T[i]$ is the progress of a task
  if $T = \emptyset$ **then return** 0
  $d \leftarrow \min_{t \in T}(\min_{1 \leq i \leq k, w_i > t} w_i - t)$
  if $d \times |T| > \beta$ **then return** 0
  `/* Jump to the next time step at which at least one task reaches a threshold */`
  return `ParSchedState`$(\beta - d \times |T|,$
  $\{T[i] + d\}_{1 \leq i \leq |T|, \exists l \ s.t. T[i]+d=w_l}, \{T[i] + d\}_{1 \leq i \leq |T|, \nexists l \ s.t. T[i]+d=w_l})$

Function `ParSchedState`(β, T_1, T_2)
  Data: The budget β
  A set T_1: $T_1[i]$ is the progress of a task that has just reached a threshold and may complete
  A set T_2: $T_2[i]$ is the progress of a task, the progress is either not equal to a threshold or it is equal to one but the task did not complete at that threshold
  if $T_1 = \emptyset$ **then**
    return `ParSchedDecision`$(\beta, T_2, \emptyset)$
  else
    Let l be such that $w_l = T_1[1]$
    `/* Either task 1 succeeds or not */`
    return $\frac{p_l}{1 - \sum_{i=1}^{l-1} p_i}$(1 + `ParSchedState`$(\beta, T_1 \setminus \{T_1[1]\}, T_2)) +$
    $(1 - \frac{p_l}{1 - \sum_{i=1}^{l-1} p_i})$ `ParSchedState`$(\beta, T_1 \setminus \{T_1[1]\}, T_2 \cup \{T_1[1]\})$

return `ParSchedDecision`$(b, \emptyset, \emptyset)$

Note that the proof also holds if the parallel algorithm is allowed to start using some new processors in the middle of the computation, or is allowed to restart a processor that it previously left idle.

Lemma 5. `ParSched` *is never worse than* `SeqSched`*, and can achieve strictly better performance on some problem instances.*

Proof. Given Lemma 4, `ParSched` is least as good as `SeqSched`. A sequential execution without preemption on a single processor is a special case of a parallel execution where the number of processors is one. Thus, `ParSched` is at least as good as `SeqSched`. Now, consider the instance $\mathcal{D} = \{(0.4, 2), (0.15, 3), (0.45, 7)\}$. The optimal expected number of tasks that can be completed on a single processor with $b = 6$ is 1.2 without preemption, whereas it is 1.236 on multiple processors. This result was obtained through the study in Section 4.1 and can be checked using Algorithms 1 and 2 on the instance. Hence, there exist instances where `ParSched` is strictly better than `SeqSched`. □

Lemma 6. `PSeqSched` *is never worse than* `ParSched`*, and can achieve strictly better performance on some problem instances.*

Proof. Given Lemma 4, `PSeqSched` is always as least as good as `ParSched`. Consider the instance $\mathcal{D} = \{(0.15, 1), (0.6, 2), (0.15, 3), (0.1, 5)\}$. The optimal expected number of tasks that can be completed on multiple processors with $b = 6$ is 2.4372 without preemption, whereas it is 2.4497 with preemption. This result is obtained by executing Algorithms 2 and 3 on the instance. □

4.3 Asymptotic behavior

In this section, we derive an asymptotically optimal strategy when letting the budget tend to infinity. Because the scheduling strategy described below is applied independently on each processor, we can assume that $p = 1$ throughout this section without loss of generality. As stated earlier, recall that we assume that there is no deadline. Note that a fixed deadline would make no sense when $b \to +\infty$ and $p = 1$. We first describe the strategy in Section 4.3.1 and show its asymptotic optimality in Section 4.3.2. Throughout this section, we are given a discrete distribution $\mathcal{D} = \{(p_i, w_i)\}_{1 \leq i \leq k}$.

4.3.1 Optimal fixed-threshold strategy

Consider a discrete distribution $\mathcal{D} = \{(p_i, w_i)\}_{1 \leq i \leq k}$. For $1 \leq i \leq k$, the i-th *fixed-threshold* strategy, or FTS_i, interrupts every unsuccessful task at threshold w_i, i.e., when the task has been executing for w_i seconds without

completing. There are k such strategies, one per threshold. Informally, our criterion to select the best one is to maximize the ratio

$$\mathcal{R} = \frac{\text{expected number of tasks completed}}{\text{budget}}$$

$$= \frac{\text{expected number of tasks completed}}{\text{total time spent}}$$

Indeed, this ratio measures the success rate per time unit, or equivalently, per budget unit (since we have unit execution speed). Formally, we would like to compute

$$\mathcal{R}_i(b) = \frac{N_i(b)}{b} \tag{1}$$

where $N_i(b)$ is the expectation of the number of tasks that are successfully completed when using strategy FTS_i that interrupts all unsuccessful tasks after w_i seconds, and proceeds until the budget b has been spent. It turns out that we can compute the limit $\mathcal{R}_i$ of $\mathcal{R}_i(b)$ when the budget b tends to infinity:

Proposition 1.

$$\lim_{b\to\infty} \mathcal{R}_i(b) = \mathcal{R}_i \stackrel{def}{=} \frac{\sum_{j=1}^{i} p_j}{\sum_{j=1}^{i} p_j w_j + (1 - \sum_{j=1}^{i} p_j) w_i}$$

Proof. Consider an execution using strategy FTS_i and with budget b. We execute n tasks during at most w_i seconds until there remains some budget, and maybe there exists a last task that is truncated due to budget exhaustion before it completes. Let b_{left} be the sum of the unused budget and of the budget spent for the truncated task (if any). The execution of the n tasks lasts $b - b_{left}$ seconds, where $0 \leq b_{left} \leq w_i$. For $1 \leq j \leq i$, let n_j denote the number of tasks that have completed successfully in exactly w_j seconds. Then $n - \sum_{j=1}^{i} n_j$ tasks have been unsuccessful and interrupted, and we have

$$b - b_{left} = n_1 w_1 + \cdots + n_i w_i + (n - \sum_{j=1}^{i} n_j) w_i.$$

Note that n, n_j for $1 \leq j \leq i$, and b_{left} are random variables here. With the notation of Equation 1, we have $N_i(b) = \mathbb{E}(\sum_{j=1}^{i} n_j)$ and we aim at showing the existence of the limit

$$\lim_{b\to\infty} \frac{N_i(b)}{b}$$

and at computing its value.

When the budget b tends to infinity, so does n, because $n \geq \left\lfloor \frac{b}{w_i} \right\rfloor$. We now show that $\frac{n_1}{n}$ converges almost surely to the value p_1: we write $\frac{n_1}{n} \stackrel{a.s.}{\to} p_1$. This means that the convergence to that limit is true, except

maybe over a set of measure zero. To see this, for the i-th task, let $X_i^{(1)}$ be the random variable whose value is 1 if the task completes in w_1 seconds, and 0 otherwise. By definition $n_1 = X_1^{(1)} + X_2^{(1)} + \cdots + X_n^{(1)}$. The $X_i^{(1)}$ are IID and have expectation $\mathbb{E}(X_i^{(1)}) = 1.p_1 + 0.(1-p_1) = p_1$, hence

$$\frac{X_1^{(1)} + X_2^{(1)} + \cdots + X_n^{(1)}}{n} \overset{a.s.}{\to} p_1$$

according to the strong law of large numbers [28, p. 212], hence the result. We prove similarly that $\frac{n_j}{n} \overset{a.s.}{\to} p_j$ for $1 \leq j \leq i$.

Then, we have:

$$\frac{\sum_{j=1}^{i} n_j}{b} = \frac{\sum_{j=1}^{i} n_j}{\sum_{j=1}^{i} n_j w_j + (n - \sum_{j=1}^{i} n_j) w_i + b_{left}}$$

$$= \frac{\sum_{j=1}^{i} \frac{n_j}{n}}{\sum_{j=1}^{i} \frac{n_j}{n} w_j + (1 - \sum_{j=1}^{i} \frac{n_j}{n}) w_i + \frac{b_{left}}{n}} \overset{a.s.}{\to} \mathcal{R}_i$$

(where $\mathcal{R}_i$ is defined in Proposition 1), because $\frac{n_j}{n} \overset{a.s.}{\to} p_j$ for $1 \leq j \leq i$, $\frac{b_{left}}{n} \overset{a.s.}{\to} 0$ (that convergence is even deterministic because b_{left} is bounded by a constant), and the finite union of sets of measure zero has measure zero. A fortiori when taking the expectations, we have deterministic convergence:

$$\mathcal{R}_i(b) = \frac{\mathbb{E}(\sum_{j=1}^{i} n_j)}{b} \to \mathcal{R}_i,$$

which concludes the proof. □

The optimal fixed-threshold strategy FTS_{opt} is defined as the strategy FTS_i whose ratio $\mathcal{R}_i$ is maximal. If several strategies FTS_i achieve the maximal ratio $\mathcal{R}_{opt}$, we pick the one with smallest w_i (to improve success rate when the budget is limited and truncation must occur). Formally:

Definition 1. *FTS_{opt} is the strategy FTS_{i_0} where $i_0 = \min_{1 \leq i \leq k}\{i \big| \mathcal{R}_i = \min_{1 \leq j \leq k} \mathcal{R}_j\}$.*

To conclude this section, we work out a little example. Consider a distribution $\mathcal{D} = \{(p_i, w_i)\}_{1 \leq i \leq 3}$ with 3 thresholds. We have

$$\mathcal{R}_1 = \frac{p_1}{w_1}, \quad \mathcal{R}_2 = \frac{p_1 + p_2}{p_1 w_1 + (1-p_1) w_2}, \text{ and}$$

$$\mathcal{R}_3 = \frac{p_1 + p_2 + p_3}{p_1 w_1 + p_2 w_2 + (1 - p_1 - p_2) w_3}$$

$$= \frac{1}{p_1 w_1 + p_2 w_2 + p_3 w_3}.$$

We pick the largest of these three values to derive FTS_{opt}.

4.3.2 Asymptotic optimality of FTS_{opt}

A scheduling strategy makes the following decisions for each task: when a new threshold is reached, and if the task is not successful at this point, decide whether either to continue execution until the next threshold, or to interrupt the task. In the most general case, these decisions may depend upon the remaining available budget. However, when the budget is large, it makes sense to restrict to strategies where such decisions are taken independently of the remaining budget, independently to past history, and either deterministically or non-deterministically but according to some fixed probabilities. We formally define such strategies as follows:

Definition 2. *A* mixed-threshold strategy $MTS(q_1, q_2, \ldots, q_{k-1})$*, where* $0 \leq q_j \leq 1$ *for* $1 \leq j \leq k-1$ *are fixed probabilities, makes the following decision when the execution of a task reaches threshold* w_i*, for* $1 \leq i \leq k-1$*, without success: it decides randomly to continue execution until the next threshold with probability* q_i*, and to interrupt the task otherwise, hence with probability* $1 - q_i$*.*

Of course, the fixed-threshold strategy FTS_i coincides with $MTS(1, \ldots, 1, 0, \ldots, 0)$ where the last 1 is in position $i-1$: $q_j = 1$ for $j < i$ et $q_j = 0$ for $j \geq i$. In this section, we prove our main result for discrete distributions:

Theorem 1. *FTS_{opt} is asymptotically optimal among all mixed-threshold strategies.*

Proof. Theorem 1 applies to any fixed number of processors p, but recall that we assume $p = 1$ w.l.o.g. in this section, because the rate per time/budget unit is computed independently for each processor. Given an arbitrary strategy $MTS(q_1, q_2, \ldots, q_{k-1})$, consider an execution with budget b and where we execute n tasks according to the strategy until the last seconds, i.e., until some instant $b - b_{left}$, where $0 \leq b_{left} \leq w_k$. As before, when the budget b tends to infinity, so does n, because $n \geq \left\lfloor \frac{b}{w_k} \right\rfloor$. In the execution, let n_i be the number of tasks whose execution has lasted w_i seconds, let m_i be the number of tasks whose execution was successful and lasted w_i seconds; scaling by n, let $\alpha_i = \frac{n_i}{n}$ and $\beta_i = \frac{m_i}{n}$ for $1 \leq i \leq k$. As in the proof of Proposition 1, using the strong law of large numbers, we prove the following:

$$
\begin{aligned}
&\beta_1 \overset{a.s.}{\to} \beta_1^\infty = p_1 \\
&\beta_2 \overset{a.s.}{\to} \beta_2^\infty = \tfrac{p_2}{1-p_1}(1 - \alpha_1^\infty) \\
&\beta_3 \overset{a.s.}{\to} \beta_3^\infty = \tfrac{p_3}{1-p_1-p_2}(1 - \alpha_1^\infty - \alpha_2^\infty) \\
&\ldots \\
&\beta_{k-1} \overset{a.s.}{\to} \beta_{k-1}^\infty = \tfrac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j}(1 - \textstyle\sum_{j=1}^{k-2} \alpha_j^\infty) \\
&\beta_k \overset{a.s.}{\to} \beta_k^\infty = \tfrac{p_k}{1-\sum_{j=1}^{k-1} p_j}(1 - \textstyle\sum_{j=1}^{k-1} \alpha_j^\infty)
\end{aligned}
$$

and

$$\begin{aligned}
&\alpha_1 \overset{a.s.}{\to} \alpha_1^\infty = p_1 + (1-p_1)(1-q_1)\\
&\alpha_2 \overset{a.s.}{\to} \alpha_2^\infty = \big(\tfrac{p_2}{1-p_1} + (1-\tfrac{p_2}{1-p_1})(1-q_2)\big)(1-\alpha_1^\infty)\\
&\alpha_3 \overset{a.s.}{\to} \alpha_3^\infty = \big(\tfrac{p_3}{1-p_1-p_2} + (1-\tfrac{p_3}{1-p_1-p_2})(1-q_3)\big)\\
&\qquad\qquad (1-\alpha_1^\infty-\alpha_2^\infty)\\
&\dots\\
&\alpha_{k-1} \overset{a.s.}{\to} \alpha_{k-1}^\infty = \big(\tfrac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j} + (1-\tfrac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j})\\
&\qquad\qquad (1-q_{k-1})\big) \times (1-\textstyle\sum_{j=1}^{k-2}\alpha_j^\infty)\\
&\alpha_k \overset{a.s.}{\to} \alpha_k^\infty = 1-\textstyle\sum_{j=1}^{k-1}\alpha_j^\infty
\end{aligned}$$

We also prove just as before that

$$\frac{b}{n} \overset{a.s.}{\to} \sum_{j=1}^{k} \alpha_j^\infty w_j$$

so that the success rate per budget unit does have the following limit when the budget tends to infinity:

$$\frac{\sum_{j=1}^{k}\beta_j}{\sum_{j=1}^{k}\alpha_j w_j} \overset{b\to\infty}{\to} \mathcal{R}(\alpha_1^\infty,\alpha_2^\infty,\dots,\alpha_{q-1}^\infty) \overset{def}{=} \frac{\sum_{j=1}^{k}\beta_j^\infty}{\sum_{j=1}^{k}\alpha_j^\infty w_j}$$

The rest of the proof is pure algebra: we have to show that the maximum value of $\mathcal{R}(\alpha_1^\infty,\alpha_2^\infty,\dots,\alpha_{q-1}^\infty)$ over all values $0 \le \alpha_j^\infty \le 1$ for $1 \le j \le k-1$, is $\mathcal{R}_{opt}$, obtained when the strategy is some FTS_i (i.e., when there exists i with $\alpha_j^\infty = 1$ if $j < i$ and $\alpha_j^\infty = 0$ if $j \ge i$). Note that below, to ease the writing, we simply use α_j instead of α_j^∞'s of the above equations, and we obtain:

$$\begin{aligned}
&\textsc{Problem } \textsc{Pb}[k](\mathcal{D}):\\
&\textsc{Maximize } \mathcal{R}(\alpha_1,\alpha_2,\dots,\alpha_{k-1}) =\\
&\qquad\qquad \frac{\sum_{i=1}^{k}\frac{p_i}{1-\sum_{j=1}^{i-1}p_j}(1-\sum_{j=1}^{i-1}\alpha_j)}{\sum_{i=1}^{k-1}\alpha_i w_i + (1-\sum_{i=1}^{k-1}\alpha_i)w_k}\\
&\textsc{Subject to } \begin{cases}
p_1 \le \alpha_1 \le 1\\
\frac{p_2}{1-p_1}(1-\alpha_1) \le \alpha_2 \le 1\\
\dots\\
\frac{p_{k-1}}{1-\sum_{j=1}^{k-2}p_j}(1-\sum_{j=1}^{k-2}\alpha_j) \le \alpha_{k-1} \le 1\\
\qquad\qquad \text{AND } \sum_{i=1}^{k} p_i \le 1
\end{cases}
\end{aligned} \tag{2}$$

We proceed by induction on k to show that the maximum value of the optimization problem $\textsc{Pb}[k](\mathcal{D})$ is $\mathcal{R}_{opt}$. Note that we do not assume that $\sum_{i=1}^{k} p_i = 1$ when stating $\textsc{Pb}[k](\mathcal{D})$ but only $\sum_{i=1}^{k} p_i \le 1$. For the base case $k = 1$, we have a single value $\frac{p_1}{w_1}$, which is the ratio $\mathcal{R}_{opt}$ of FTS_1.

For the case $k = 2$, we have a single variable α_1, where $p_1 \leq \alpha_1 \leq 1$, and $\mathcal{R}(\alpha_1) = \frac{p_1 + \frac{p_2}{1-p_1}(1-\alpha_1)}{\alpha_1 w_1 + (1-\alpha_1) w_2}$. We note that the derivative of the function $x \to f(x) = \frac{ax+b}{cx+d}$ has constant sign (that of $ad - bc$); hence, the maximum of $\mathcal{R}(\alpha_1)$ is obtained for one of the two bounds, either $\alpha_1 = p_1$, with value $\frac{p_1+p_2}{p_1 w_1 + (1-p_1) w_2}$, or $\alpha_1 = 1$, with value $\frac{p_1}{w_1}$. The first value is the ratio $\mathcal{R}_2$ of FTS_2, and the second value is the ratio $\mathcal{R}_1$ of FTS_1, which concludes the proof for $k = 2$.

Assume that we have shown the result for $\textsc{Pb}[k'](\mathcal{D}')$ for $2 \leq k' \leq k-1$ and all distributions $\mathcal{D}$' with k' thresholds, and consider the problem $\textsc{Pb}[k](\mathcal{D})$. First we fix the values of α_j, $1 \leq j \leq k-2$, and view $\mathcal{R}(\alpha_1, \alpha_2, \ldots, \alpha_{k-1})$ as a function of α_{k-1}. It is again of the form $x \to f(x) = \frac{ax+b}{cx+d}$; hence, the maximum is obtained for one of the two bounds, either $\alpha_{k-1} = \frac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j}(1 - \sum_{j=1}^{k-2} \alpha_j)$, or $\alpha_{k-1} = 1 - \sum_{j=1}^{k-2} \alpha_j$.

First case If $\alpha_{k-1} = \frac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j}(1 - \sum_{j=1}^{k-2} \alpha_j)$, then, $1 - \sum_{j=1}^{k-1} \alpha_j = (1 - \frac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j})(1 - \sum_{j=1}^{k-2} \alpha_j)$. Hence, $\frac{p_k}{1-\sum_{j=1}^{k-1} p_j}(1 - \sum_{j=1}^{k-1} \alpha_j) = \frac{p_k}{1-\sum_{j=1}^{k-2} p_j}(1 - \sum_{j=1}^{k-2} \alpha_j)$. Thus

$$\mathcal{R}(\alpha_1, \alpha_2, \ldots, \alpha_{k-2}, \frac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j}(1 - \sum_{j=1}^{k-2} \alpha_j)) =$$

$$\frac{\sum_{i=1}^{k-1} \frac{p_i}{1-\sum_{j=1}^{i-1} p_j}(1-\sum_{j=1}^{i-1} \alpha_j) + \frac{p_k}{1-\sum_{j=1}^{k-1} p_j}(1-\sum_{j=1}^{k-1} \alpha_j)}{\sum_{i=1}^{k-2} \alpha_i w_i + (1-\sum_{j=1}^{k-2} \alpha_j) \frac{p_{k-1} w_{k-1} + (1-\sum_{j=1}^{k-1} p_j) w_k}{1-\sum_{j=1}^{k-2} p_j}}.$$

Consider the distribution $\mathcal{D}' = \{p'_i, w'_i\}_{1 \leq i \leq k-1}$ such that $p'_i = p_i$ and $w'_i = w_i$ for $1 \leq i \leq k-2$, $p'_{k-1} = p_{k-1} + p_k$ and $w'_{k-1} = \frac{p_{k-1} w_{k-1} + (1-\sum_{j=1}^{k-1} p_j) w_k}{1-\sum_{j=1}^{k-2} p_j}$. The distribution $\mathcal{D}$' has $k-1$ thresholds. The optimization problem $\textsc{Pb}[k-1](\mathcal{D}')$ writes

$$\textsc{Maximize } \mathcal{R}'(\alpha'_1, \alpha'_2, \ldots, \alpha'_{k-2}) = \frac{\sum_{i=1}^{k-1} \frac{p'_i}{1-\sum_{j=1}^{i-1} p'_j}(1-\sum_{j=1}^{i-1} \alpha'_j)}{\sum_{i=1}^{k-2} \alpha'_i w'_i + (1-\sum_{i=1}^{k-2} \alpha'_i) w'_{k-1}}.$$

$$\textsc{Subject to } \frac{p'_i}{1-\sum_{j=1}^{i-1} p'_j}(1 - \sum_{j=1}^{i-1} \alpha'_j) \leq \alpha'_i \leq 1, \quad \forall i \in \{1, \ldots, k-2\} \quad \textsc{and } \sum_{i=1}^{k-1} p'_i \leq 1$$

Replacing p'_{k-1} and w'_{k-1} by their values, we see that $\textsc{Pb}[k](\mathcal{D})$ when $\alpha_{k-1} = \frac{p_{k-1}}{1-\sum_{j=1}^{k-2} p_j}(1 - \sum_{j=1}^{k-2} \alpha_j)$ reduces to $\textsc{Pb}[k-1](\mathcal{D}')$. By induc-

tion hypothesis, $\textsc{Pb}[k-1](\mathcal{D}')$ achieves its maximum for some fixed-threshold strategy FTS'_i, where $1 \leq i \leq k-1$. The task-to-budget ratios $\mathcal{R}'_i$ for $\mathcal{D}$' are the following:

- $\mathcal{R}'_i = \mathcal{R}_i$ for $1 \leq i \leq k-2$
- $\mathcal{R}'_{k-1} = \frac{\sum_{j=1}^{k-1} p'_j}{\sum_{j=1}^{k-2} p'_j w'_j + (1-\sum_{j=1}^{k-2} p'_j) w'_{k-1}} = \frac{\sum_{j=1}^{k-2} p_j + p_{k-1} + p_k}{\sum_{j=1}^{k-2} p_j w_j + p_{k-1} w_{k-1} + (1-\sum_{j=1}^{k-1} p_j) w_k} = \mathcal{R}_k.$

This is the desired result and concludes the analysis for the first case.

Second case If $\alpha_{k-1} = 1 - \sum_{j=1}^{k-2} \alpha_j$, then

$$\mathcal{R}(\alpha_1, \alpha_2, \ldots, \alpha_{k-2}, 1 - \sum_{j=1}^{k-2} \alpha_j) =$$

$$\frac{\sum_{i=1}^{k-1} \frac{p_i}{1-\sum_{j=1}^{i-1} p_j} (1 - \sum_{j=1}^{i-1} \alpha_j)}{\sum_{i=1}^{k-2} \alpha_i w_i + (1 - \sum_{j=1}^{k-2} \alpha_j) w_{k-1}}.$$

Consider the distribution $\mathcal{D}' = \{p'_i, w'_i\}_{1 \leq i \leq k-1}$ such that $p'_i = p_i$ and $w'_i = w_i$ for $1 \leq i \leq k-1$. The distribution $\mathcal{D}$' has $k-1$ thresholds. The optimization problem $\textsc{Pb}[k](\mathcal{D})$ when $\alpha_{k-1} = 1 - \sum_{j=1}^{k-2} \alpha_j$ directly reduces to $\textsc{Pb}[k-1](\mathcal{D}')$. By induction hypothesis, $\textsc{Pb}[k-1](\mathcal{D}')$ achieves its maximum for some fixed-threshold strategy FTS'_i, where $1 \leq i \leq k-1$. The task-to-budget ratios for $\mathcal{D}$' are the same as for $\mathcal{D}$: $\mathcal{R}'_i = \mathcal{R}_i$ for $1 \leq i \leqq k-2$. This is the desired result and concludes the analysis for the second case.

Altogether, we have solved the optimization problem $\textsc{Pb}[k](\mathcal{D})$. This concludes the proof of the theorem. □

5 Continuous distributions

In this section, we build upon the previous results and deal with continuous distributions. We do assume to have a fixed budget and a deadline. Thus, in contrast to Section 4, the distribution $\mathcal{D}$ is now continuous and has expected value μ_D and variance σ_D^2. Let $F(x)$ be its cumulative distribution function and $f(x)$ its probability density function. The objective remains to execute as many tasks as possible given a budget b, a deadline d and a potentially unlimited number of processors.

We start by designing several heuristics in Section 5.1 and then we assess their efficiency through experiments in Section 5.2. The code and scripts used for the simulations and the data analysis are publicly available online [9].

5.1 Heuristics

We present below different heuristics, among which an extension of the asymptotically optimal greedy strategy of Section 4.3 to the continuous case. In all cases, we enroll $\lceil \frac{b}{d} \rceil$ machines. The rationale for this choice is that this is the maximum number of machines that can work in parallel and continuously, up to the deadline.

We have three main classes of heuristics:

- MeanVariance(x) is the family of heuristics that kill a task as soon as its execution time reaches $\mu_D + x\sigma_D$, where x is some positive or negative constant.
- Quantile(x) is the family of heuristics that kill a task when its execution time reaches the x-quantile of the distribution $\mathcal{D}$ with $0 \leq x \leq 1$.
- OptRatio is the heuristic inspired by the asymptotically optimal strategy for discrete distributions. OptRatio interrupts all (unsuccessful) tasks at time

$$l = \arg\max_{l} \mathcal{R}(l)$$

where

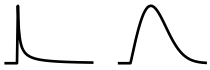
$$\mathcal{R}(l) = \frac{F(l)}{\int_0^l x f(x)dx + l(1 - F(l))}.$$

The idea behind OptRatio is that it maximizes the ratio of the probability of success (namely $F(l)$) to the expected amount of budget spent for a single task when the task is interrupted at time l (i.e., $\int_0^l x f(x)dx$ for the cases when the task terminates sooner than l and $\int_l^\infty l f(x)dx = l(1 - F(l))$ otherwise). This is a continuous extension of the approach proposed in Section 4.3, and we expect OptRatio to perform well for large budgets.

We now analyze OptRatio with some classical probability distributions defined on nonnegative values (task execution times need to be nonnegative). For the exponential distribution, which is memoryless, $\mathcal{R}(l) = \lambda$ where λ is the rate of the distribution. In this case, any l can be chosen and the tasks may be interrupted at any moment with OptRatio without modifying the performance. For the uniform distribution (between a and b), $\mathcal{R}(l) = 2\frac{l-a}{-l^2+2bl-a^2}$, which takes its maximum value for $l = b$ ($\mathcal{R}(b) = \frac{2}{a+b}$). In this case, tasks should never be interrupted to maximize performance. We established these results for exponential and uniform distributions through simple algebraic manipulations.

In addition to the exponential and uniform distributions, Table 1 presents other standard distributions. For these distributions, we provide some code [9] to numerically compute the optimal time l at which tasks should be interrupted. Note that there exist many relations between probability distributions. For instance, the beta distribution with both shape parameters equal to one is the same as the uniform distribution, whereas it has a U-shape with

Table 1: Probability distributions with their Probability Distribution Function (PDF), parameters and density graph. Supports are $[0, \infty)$ for all distributions except for Uniform, where it is $[a, b]$ and Beta, where it is $[0, 1]$. Note that $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$.

Name	PDF	Parameters	Density
Uniform	$\frac{1}{b-a}$	$-\infty < a < b < \infty$	
Exponential	$\lambda e^{-\lambda x}$	rate $\lambda > 0$	
Half-normal	$\frac{\sqrt{2}}{\theta\sqrt{\pi}} e^{-\frac{x^2}{2\theta^2}}$	scale $\theta > 0$	
Lognormal	$\frac{1}{x\beta\sqrt{2\pi}} e^{-\frac{(\log(x)-\alpha)^2}{2\beta^2}}$	$\alpha \in (-\infty, \infty)$ and $\beta > 0$	
Beta	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	shape $\alpha > 0$ and shape $\beta > 0$	
Gamma	$\frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$	shape $k > 0$ and scale $\theta > 0$	
Weibull	$\frac{k}{\theta^k} x^{k-1} e^{-(\frac{x}{\theta})^k}$	shape $k > 0$ and scale $\theta > 0$	
Inverse-gamma	$\frac{\theta^k}{\Gamma(k)} x^{-k-1} e^{-\frac{\theta}{x}}$	shape $k > 0$ and scale $\theta > 0$	

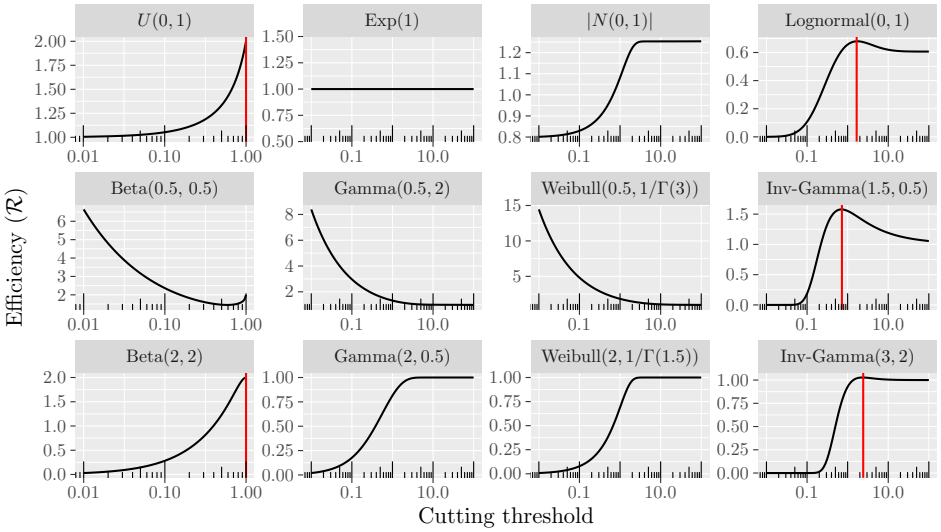


Figure 1: Efficiency (ratio $\mathcal{R}$ of number of tasks successfully executed per budget unit) for different probability distributions. Some distributions have an optimal finite cutting threshold depicted with a vertical red line.

both equal to 0.5, and a bell-shape with both equal to 2. Also, the exponential distribution is a special case of the gamma and Weibull distributions when their shape parameter is one.

Figure 1 shows how $\mathcal{R}(l)$ varies as a function of the cutting threshold l, for the probability distributions shown in Table 1. Recall that OptRatio will select the threshold l for which $\mathcal{R}(l)$ is maximum. For instance, this threshold is $l = 1$ for the uniform distribution, meaning that we should never interrupt any task. The threshold can be any value of l for the exponential distribution, and this is due to the memory less property: we can interrupt a task at any moment, without any expected consequence. The threshold is $l = \infty$ for the half-normal distribution, meaning again that we should never interrupt any task, just as for uniform distributions. Note that the expected value of all distributions is not the same overall, because we use standard parameters in Figure 1, hence ratio values are not comparable across distributions.

We remark that the lognormal distribution, which presents a fast increase followed by a slow decrease with an heavy tail, exhibits an optimal cutting threshold during the execution of a task: on Figure 1, we see that the optimal threshold is $l \approx 1.73$ (we computed this value numerically) for the distribution Lognormal$(0, 1)$. We make a similar observation for the inverse-gamma distributions, where the optimal threshold is $l \approx 0.7$ for Inv-Gamma$(1.5, 0.5)$ and $l \approx 2.32$ for Inv-Gamma$(3, 2)$. These lognormal and inverse-gamma distributions share the following properties: the density

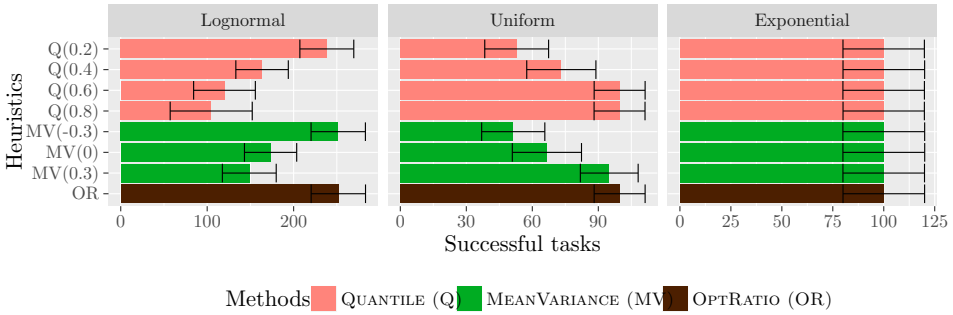


Figure 2: Number of successfully executed tasks for each heuristic with three distributions (Lognormal, Uniform, Exponential) of same expected value $\mu = 1$ and standard deviation $\sigma = 3$, with a budget and deadline $b = d = 100$ (which means that a single machine is enrolled). Each heuristic is run 100,000 times for each scenario. The error bars are computed with the mean plus/minus two standard deviations of the number of successes. The lognormal distribution has parameters $\alpha \approx -1.15$ and $\beta \approx 1.52$ to have an expected value $\mu = 1$ and a standard deviation $\sigma = 3$, and the optimal cutting threshold for OptRatio is $l \approx 0.1$). The exponential distribution has shape $\lambda = 1$ and the cutting threshold is arbitrarily set to $l = 2$. The uniform distribution has parameters $a = 0$ and $b = 2$, and the cutting threshold is $l = 2$.

is close to zero for small costs and has a steep increase. On the contrary, the bell-shape beta distribution Beta$(2, 2)$ has a small density for small costs but does not have a steep increase, and tasks should never be interrupted (in other words, the optimal cutting threshold is $l = 1$ for Beta$(2, 2)$).

Finally, we observe that three distributions are the most efficient when the cutting threshold tends to zero (Beta$(0.5, 0.5)$, Gamma$(0.5, 2)$ and Weibull$(0.5, 1/\Gamma(3))$). We point out that it is unlikely that such distributions would model actual execution times in practice.

5.2 Experiments

The following experiments make use of three standard distributions: exponential, uniform, and lognormal. The first two distributions are very simple and easy to use, while the latter has been advocated to model file sizes [13], and we assume that task costs could naturally obey this distribution too. Moreover, the lognormal distribution is positive, it has a tail that extends to infinity and the logarithm of the data values are normally distributed. Also, this distribution leads to a non-trivial cutting threshold, contrarily to exponential (interrupt anywhere) or uniform (never interrupt), thereby allowing for a complete assessment of our approach.

Figure 2 shows the number of successfully executed tasks for each heuristic with three distributions (lognormal, uniform, exponential) of same expected value $\mu = 1$ and standard deviation $\sigma = 3$, with a budget and deadline $b = d = 100$. Note that to ensure a given expected value and standard deviation for the lognormal distribution, we set its parameters as follows: $\alpha = \log(\mu) - \log(\sigma^2/\mu^2 + 1)/2$ and $\beta = \sqrt{\log(\sigma^2/\mu^2 + 1)}$. Note also that using a standard deviation $\sigma = 3$ corresponds to a high level of heterogeneity. To see this intuitively, take a discrete distribution with 11 equally probable costs, 10 of value 0.1 and 1 of value 10: its expected value is $\mu = 1$ while its standard deviation is $\sigma \approx 2.85$. Finally, we note that Figure 2 confirms that tasks with exponentially distributed costs can be interrupted at any time and that tasks with uniformly distributed costs should never be interrupted.

Next, we focus on the lognormal distribution. First, in Figure 3, we assess the impact of three important parameters: the standard deviation, the budget and the deadline, respectively. The expected value is always $\mu = 1$. By default, the standard deviation is $\sigma = 3$, and the budget and deadline are set to 100 ($b = d = 100$), which means that a single machine is enrolled. When we vary the standard deviation (first row in Figure 3), we keep $b = d = 100$. When we vary the budget (second row of in Figure 3), we maintain the equality $b = d$. When we vary the deadline (third row of in Figure 3), we keep $b = 100$, hence more VMs are enrolled (10 VMs when $d = 10$ and 100 VMs when $d = 1$). Each heuristic is run 100,000 times for each scenario. The error bars represent an interval from the mean of two standard deviations of the number of successes. For a normal distribution, this means that more than 95% of the values are in this interval. Note that the subfigures with $\sigma = 3$, $b = 100$ and $d = 100$ in Figure 3 are all the same as the subfigure with the lognormal distribution in Figure 2.

On Figure 3, we see that the higher the standard deviation, the larger the gain of every approach. With a low standard deviation, all approaches perform similarly. Increasing the budget tends to decrease the variability when running several times the same approach (the error bars are narrower with large budgets, which makes the approaches more predictable). This is an consequence of the law of large numbers. However, the expected efficiency (around 2.5 tasks per unit of time) remains similar even for a low budget of 30. Finally, decreasing significantly the deadline prevents some strategies from letting tasks run a long time. Long running tasks are then forced to be interrupted early, which is similar to the behavior of the more efficient approaches.

In all tested situations, the OptRatio algorithm with the optimal threshold achieved the best results.

Finally, Figure 4 depicts the efficiency of OptRatio with small deadlines. Even though our approach extends a strategy that is asymptotically optimal when both the budget and the deadline are large, it does perform well with small deadlines, as long as d is not lower than the cutting thresh-

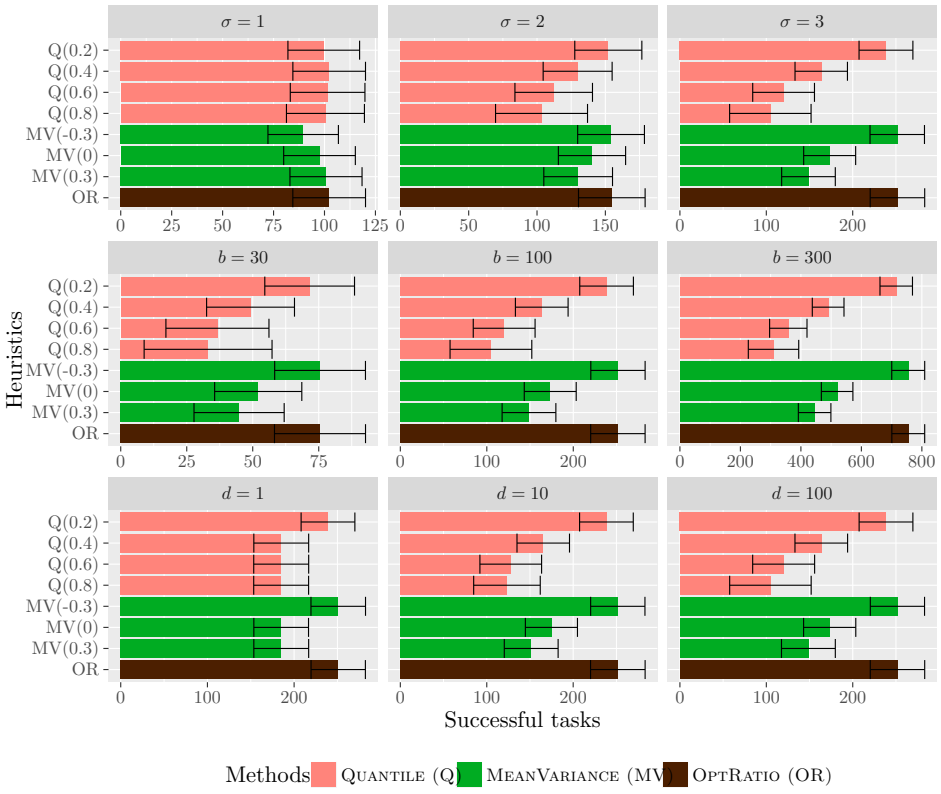


Figure 3: Number of successfully executed tasks for each heuristic, with lognormal costs and expected value $\mu = 1$. Unless otherwise specified, the standard deviation is $\sigma = 3$, and the budget and deadline are $b = d = 100$. Each heuristic is run 100,000 times for each scenario. The error bars are computed with the mean plus/minus two standard deviations of the number of successes. The lognormal distribution has parameters $\alpha \approx -1.15$ and $\beta \approx 1.52$ by default (to have $\mu = 1$ and $\sigma = 3$) (the cutting threshold for OptRatio is $l \approx 0.1$). They are $\alpha \approx -0.35$ and $\beta \approx 0.83$ when $\sigma = 1$ ($l \approx 2.1$) and $\alpha \approx -0.8$ and $\beta \approx 1.27$ when $\sigma = 1$ ($l \approx 0.34$).

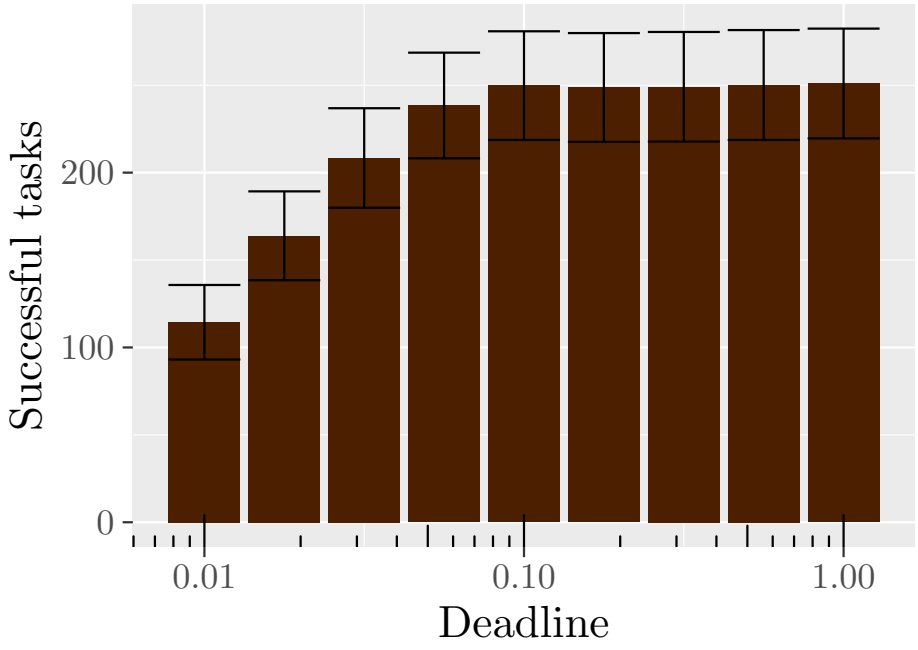


Figure 4: Number of successfully executed tasks for OptRatio with a budget $b = 100$ and optimal cutting threshold $l \approx 0.1$. OptRatio is run 100,000 times for each deadline. The error bars are computed with the mean plus/minus two standard deviations of the number of successes. The lognormal distribution has parameters $\alpha \approx -1.15$ and $\beta \approx 1.52$ to have an expected value $\mu = 1$ and a standard deviation $\sigma = 3$.

old. In the settings of Figure 4, where the average execution time of a task is equal to 1, this means that as soon as the deadline is equal to 0.1 OptRatio achieves its asymptotic performance! (The reader can compare the performance of OptRatio for deadlines of 200 and 0.1 on Figures 2 and 4.) Finally note that on Figure 4, $b = 100$ and that, therefore, OptRatio uses 1,000 processors. This confirms that neither the budget, nor the deadline need to be large for OptRatio to reach its best efficiency, and that this heuristic is extremely robust.

6 Conclusion

This paper deals with scheduling strategies to successfully execute the maximum number of a bag of stochastic tasks on VMs (Virtual Machines) with a finite budget and under a deadline constraint. We first focused on the problem instance with discrete probability distributions and no deadline. We proposed three optimal dynamic programming algorithms for different scenarios, depending upon whether tasks may be preempted or not, and whether multiple VMs may be enrolled or only a single one. We also introduced an asymptotically optimal method that computes a cutting threshold that is independent of the remaining budget. Then we extended this approach to the continuous case and with deadline. We designed OptRatio, an efficient heuristic which we validated through simulations with classical distributions such as exponential, uniform, and lognormal. Tests with several values of the deadline, leading to enroll different numbers of VMs, also confirm the relevance and robustness of our proposition.

Future work will be dedicated to considering heterogeneous tasks (still with stochastic costs), as well as heterogeneous VMs. Typically, cloud providers provide a few different categories of VM with different computer power and nominal cost, and it would be interesting (albeit challenging) to extend our study to such a framework. Another interesting direction would be to take into account start-up costs when launching a VM, thereby reducing the amount of parallelism, because fewer VMs will likely be deployed.

References

[1] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[2] M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of qos in real-time databases supporting imprecise computations. *IEEE Trans. Computers*, 55(3):304–319, 2006.

[3] V. Arabnejad, K. Bubendorfer, and B. Ng. Budget distribution strategies for scientific workflow scheduling in commercial clouds. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 137–146, Oct 2016.

[4] M. U. Bokhari, Q. Makki, and Y. K. Tamandani. A survey on cloud computing. In D. M. V. Aggarwal, V. Bhatnagar, editor, *Big Data Analytics*, volume 654 of *Advances in Intelligent Systems and Computing*. Springer, 2018.

[5] G. Buttazzo. Handling overload conditions in real-time systems. In S. M. Babamir, editor, *Real-Time Systems, Architecture, Scheduling, and Application*, chapter 7. InTech, Rijeka, 2012.

[6] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8):1011 – 1026, 2011.

[7] R. N. Calheiros and R. Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1787–1796, July 2014.

[8] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert. Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms. In *27th International Heterogeneity in Computing Workshop HCW 2013*. IEEE Computer Society Press, 2018.

[9] L.-C. Canon, A. Kong Win Chang, F. Vivien, and Y. Robert. Code for scheduling independent stochastic tasks under deadline and budget constraints, June 2018. `https://doi.org/10.6084/m9.figshare.6463223.v2`.

[10] H. Casanova, M. Gallet, and F. Vivien. Non-clairvoyant scheduling of multiple bag-of-tasks applications. In *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference*, pages 168–179, 2010.

[11] J. Y. Chung, J. W. S. Liu, and K. J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Computers*, 39(9):1156–1174, 1990.

[12] H. M. Fard, R. Prodan, and T. Fahringer. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments.

IEEE Transactions on Parallel and Distributed Systems, 24(6):1203–1212, June 2013.

[13] D. Feitelson. Workload modeling for computer systems performance evaluation. *Version 1.0.3*, pages 1–607, 2014.

[14] W. Feng and J. W. S. Liu. An extended imprecise computation model for time-constrained speech processing and generation. In *[1993] Proceedings of the IEEE Workshop on Real-Time Applications*, pages 76–80, May 1993.

[15] T. S. Ferguson. *Optimal stopping and applications*. UCLA Press, 2008.

[16] Y. Gao, Y. Wang, S. K. Gupta, and M. Pedram. An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sept. 2013.

[17] A. Grekioti and N. V. Shakhlevich. Scheduling bag-of-tasks applications to optimize computation time and cost. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics. PPAM 2013.*, volume 8385 of *Lecture Notes in Computer Science*. Springer, 2014.

[18] H. Hassan, J. Simó, and A. Crespo. Flexible real-time mobile robotic architecture based on behavioural models. *Engineering Applications of Artificial Intelligence*, 14(5):685 – 702, 2001.

[19] E. Hwang and K. H. Kim. Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 130–138, Washington, DC, USA, 2012. IEEE Computer Society.

[20] F. Jumel and F. Simonot-Lion. Management of anytime tasks in real time applications. In *XIV Workshop on Supervising and Diagnostics of Machining Systems*, Karpacz/Pologne, 2003. Colloque avec actes et comité de lecture. internationale.

[21] H. Kobayashi and N. Yamasaki. Rt-frontier: a real-time operating system for practical imprecise computation. In *Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 255–264, May 2004.

[22] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. In A. M.

van Tilborg and G. M. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 203–249, Boston, MA, 1991. Springer US.

[23] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang. A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform. *The International Journal of High Performance Computing Applications*, 24(4):445–456, 2010.

[24] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11. IEEE, Nov 2012.

[25] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. *Future Generation Computer Systems*, 48:1–18, 2015.

[26] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 41–48. IEEE, Oct. 2010.

[27] J. Meng, S. Chakradhar, and A. Raghunathan. Best-effort parallel execution framework for recognition and mining applications. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–12, May 2009.

[28] R. Nelson. *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer Science & Business Media, New York, 1995.

[29] A. M. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 351–359, Nov. 2010.

[30] A.-M. Oprescu, T. Kielmann, and H. Leahu. Budget estimation and control for bag-of-tasks scheduling in clouds. *Parallel Processing Letters*, 21(02):219–243, 2011.

[31] A. M. Oprescu, T. Kielmann, and H. Leahu. Stochastic tail-phase optimization for bag-of-tasks execution in clouds. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 204–208. IEEE, Nov. 2012.

[32] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pages 858–865, May 2014.

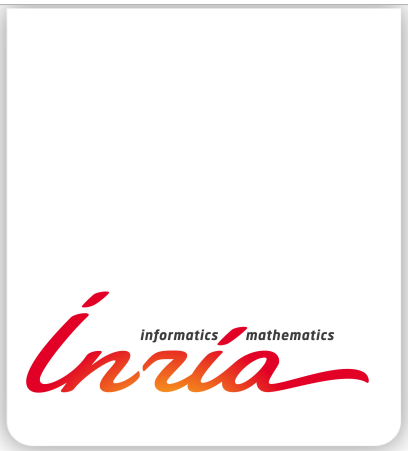
[33] S. Singh and I. Chana. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems*, 49(3):1005–1069, Dec. 2016.

[34] S. Singh and I. Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14(2):217–264, June 2016.

[35] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 155–162. IEEE, July 2011.

[36] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. *Future Generation Computer Systems*, 28(1):58 – 65, 2012.

[37] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *IEEE Transactions on Cloud Computing*, 3(2):169–181, April 2015.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399